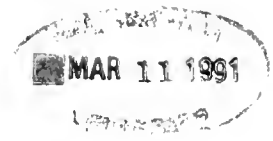

Solving Parametric Design Problems Requiring Configuration Choices

by
Rajan Ramaswamy
Karl Ulrich
Norimasa Kishi

WP #3248-91-MSA February 1991

Solving Parametric Design Problems Requiring Configuration Choices

by
Rajan Ramaswamy
Karl Ulrich
Norimasa Kishi

WP #3248-91-MSA February 1991

Solving Parametric Design Problems Requiring Configuration Choices

Rajan Ramaswamy
Karl Ulrich*
Massachusetts Institute of Technology

Norimasa Kishi
Minoru Tomikashi
Nissan Motor Company Ltd.

Abstract

Many design tasks involve selection from a set of configurations followed by parametric optimization of the chosen configuration. The models used for these tasks tend to be large, non-linear and involve both discrete and continuous variables. It is rarely possible to use any single formal algorithm to solve these problems and as a result there are very few tools to help designers solve such problems. We believe computer environments that allow flexible access to a varied set of computer tools will help designers rapidly generate high quality solutions. We demonstrate our arguments on a design problem taken from a commercial auto manufacturer, propose a framework for dealing with the general class of problems and describe a preliminary implementation of a novel design system that integrates math programming, knowledge-based and graph theoretic tools.
Keywords: Knowledge-based systems, mathematical programming, hybrid systems, graph theory, design.
Number of Words (text): 5800, **(abstract, appendices etc)**: 1100

1 Introduction

This paper explores decision support tools for engineers who deal with complex input-output models. An input-output model is used to compute a set of performance parameters from a set of input parameters, based on a set of equations (see Figure 1). The

*Direct all correspondence to this author at MIT Rm E53-389, Cambridge MA 02139, (617)253-0487, ulrich@ai.mit.edu.

design task is to specify a set of inputs that yield outputs satisfying certain criteria. These inputs typically include a configuration choice and values for the various parameters associated with that configuration. This is a hard problem when the number of configurations is large and the models are nonlinear.

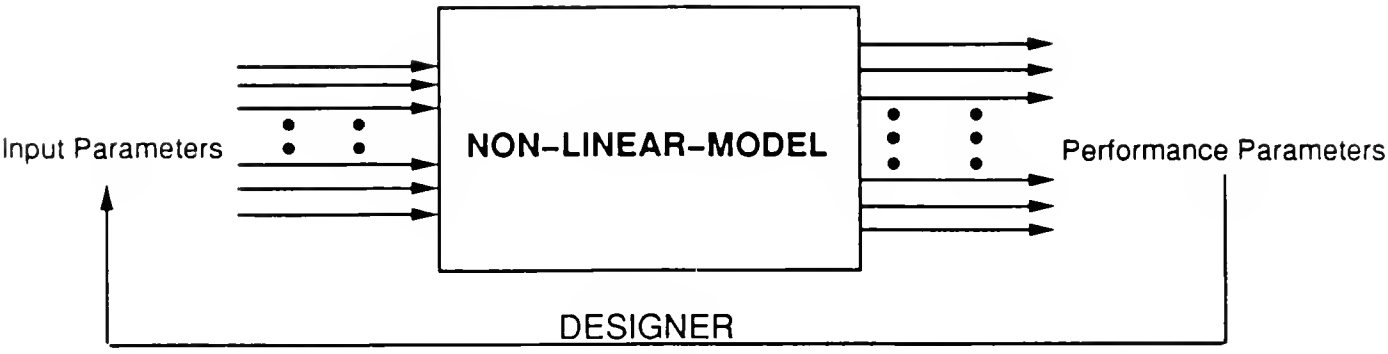

Figure 1: Input-Output Model

This work

- Develops new methods for decomposing or simplifying complex problems.
- Proposes a novel framework which combines currently monolithic tools into a usable, integrated system. To this end we catalog, organize and evaluate existing techniques.
- Addresses practical implementation issues by describing a prototype integrated system used to solve a suitable problem of industrial importance.

1.1 Sample Domain: Preliminary Design of Automobiles

Our example is drawn from a commercial system used for preliminary design of automobiles. In the example, automobile performance is quantified as a set of 19 performance parameters such as fuel economy and braking distance. The input parameters used to calculate the performance parameters are classified as

- *Concept* parameters. These are prespecified to the designer and cannot be changed. For example, the transmission may be required to be automatic.
- *Database* parameters. These are looked up from databases and therefore also cannot be changed. For example, displacement of the engine in the current model of car.
- *Design* parameters. These can be varied by the designer. Design parameters are either configuration related and involve discrete choices, e.g. choice of a turbocharger from the set (none, twin, double, intercooler) or are continuous variable choices e.g. engine displacement in the range 1000cc to 3000cc.

An outline of the currently used design process is as follows. First, the designer receives a list of target values for the performance parameters from the marketing organization along with a list of values for the concept parameters. A current database is

available for lookup of database parameter values. Then, the designer tries to achieve the target values by manipulating the inputs to the computer model (see Figure 1). If this cannot be done, the targets are modified through a negotiation process with marketing.

In current practice, the designer receives no support from the computer system other than the ability to recompute the outputs once the inputs are changed. Using this methodology, experienced users can achieve acceptable results in a reasonable time (usually a day or two) but are usually unable to generate multiple solutions. Some features the users desire in an improved system are,

- Automatic computation of several sets of inputs that achieve the desired targets.
- Use of an explicit optimality criterion.
- Faster re-calculation of outputs when inputs are changed.

In addition, there is interest in developing better computer-based data representations for this application. Most current representations trade off efficiency for maintainability and flexibility. None of these properties can be neglected if the system is to be successful.

type	actual	sample
input parameters	531	75
design parameters	200	44
performance parameters	19	6
equations	331	50

Table 1: Model Size Description

1.2 Technical Challenges

The main technical hurdles to satisfying the above specifications are,

1. Complexity. The problem requires optimization with multiple objectives, nonlinear and integer constraints, nonlinear objective function and no obvious special structure. The class of mixed integer nonlinear problems is provably NP-complete[28].

2. Size. The size of the problem (Table 1) is comparable to the largest problems described in the literature.

3. Formulation. Formulation of this problem as a mathematical program and verification of that formulation is non-trivial due to the nature and number of the constraints.

1.3 Sample I/O Model

Because of size considerations (see Table 1), it was not possible to study the entire model for an automobile. Instead, we conducted our work on a 30 percent sample of the complete model. Table 1 shows a size comparison.

Figure 2 shows the graph of an input-output model used to compute two performance parameters. The examples given in this paper refer only to this example. The equations corresponding to Figure 2 are explained in Section 8.

1.4 Organization of the paper

In Section 2 we describe how graph representations and algorithms can be applied to reveal problem simplifications and decompositions. Section 3 covers the use of knowledge-based systems to make discrete choices. In Section 4 we discuss how nonlinear programming can be applied once the discrete variables are fixed and also discuss how mixed integer nonlinear programming can be used to determine suitable values for both the continuous and discrete variables in a single step. The final section describes how the different algorithms described can be usefully combined and accessed from a single environment. Appendix 1 is a review of the relevant literature and Appendix 2 contains the complete equations corresponding to Figure 2.

2 Graph-Based Problem Simplification

Conventional wisdom on solving large mixed integer nonlinear problems is that physical insight is useful in simplifying and constraining a problem. Often, this insight is the difference between getting a solution and dismissing the problem as too complex to solve. However, we are not aware of any efforts to study the nature of these insights or to perceive common features between different fields. We would like to know what insight amounts to in a mathematical sense. Is insight related to notions of independence, monotonicity and irrelevance or has it no known mapping into mathematics? One possibility, is that this insight is the ability to locate minimally coupled portions of the design which may be examined independently. In this section, we describe some algorithms that can be used to simplify equation trees of the type shown in Figure 2, and hence automate this form of problem simplification.

2.1 Tools for Managing Complexity

We present a few examples of the types of information that can be automatically derived from a constraint network. We have implemented these methods as LISP functions that can be called on the network.

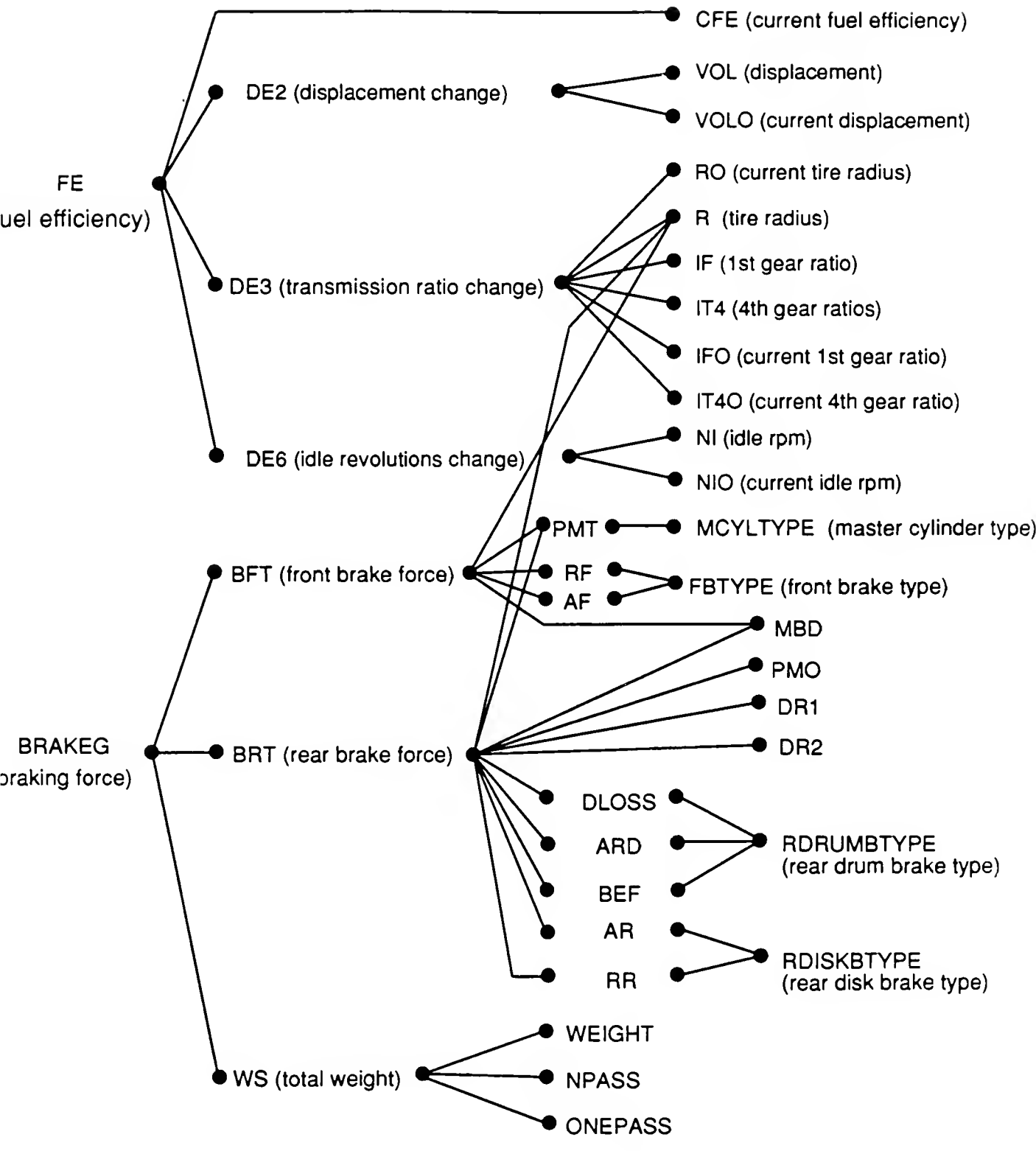

Figure 2: Dependency graph for fuel-efficiency and braking-force performance parameters. (Physical significance of selected variables is shown in parentheses, "current" values refer to existing model of car)

The first class is composed of functions that do not do any searching and gather useful information only by traversing links within the network. They help designers by automatically deriving high level information about the network. Some such functions are as follows:

- *Get-General-Network-Info* retrieves general information about network. For example, when this is run on the network in Figure 2 it prints the following information.

Tree Statistics

Number of Performance Parameters:	2
Number of Parameters (including intermediate):	36
Number of Input Parameters:	22

Performance Parameter Statistics

Performance Parameter	BRAKEG
Number of Intermediate Parameters	11
Number of Input Parameters	12
Exclusive Input Parameters	(RDISKBTYPE RDRUMBTYPE FBTYPE MCYLTYPE MBD PMO DR1 DR2 WEIGHT NPASS ONEPASS)

Performance Parameter	FE
Number of Intermediate Parameters	3
Number of Input Parameters	11
Exclusive Input Parameters	(CFE VOL VOLO RO IF IT4 IFO IT4O NI NIO)

Intermediate parameters are those that are neither roots nor leaves of the tree. The exclusive input parameters are leaves of the tree that are connected to one and only one performance parameter. Simple predicates are available to tell if the type of an input parameter is concept, database or design.

- *Get-All-Input-Parameters* retrieves all the input parameters affecting a particular performance parameter. For example, when called on the braking distance parameter, it returns RDISKBTYPE, RDRUMBTYPE, FBTYPE, MCYLTYPE, R, MBD, PMO, DR1, DR2, WEIGHT, NPASS, ONEPASS.
- *Get-Exclusive-Design-Parameters* retrieves input parameters that affect only the named performance parameter (and no other) and are also design parameters. For example, the results of calling this function on BRAKEG (braking force) are RDISKBTYPE, RDRUMBTYPE, FBTYPE and MCYLTYPE.
- *Get-Common-Input-Parameters* searches for and retrieves input parameters shared by two performance parameters. For example, the function can tell the designer that the FE (fuel efficiency) and the BRAKEG (braking force) parameters have only a single common design variable, namely R (tire radius). Other pairs such as starting acceleration and cruising acceleration (two other performance parameters) may be more strongly coupled and have more than one common input.
- *Get-Related-Performance-Parameters* tells the designer which performance-parameters

would be affected by changing the value of a particular input parameter. For example, a change in R (tire radius) will affect both FE (fuel efficiency) and BRAKEG (braking force) but a change in MCYLTYPE (master cylinder type) will affect only BRAKEG and will not change FE.

These functions are a useful addition to the system because both experienced designers and new users can use them to advantage. For example,

- A designer can explicitly recall and confirm dependencies or even discover new ones. This is particularly useful in situations where models are constantly being updated. New users can use these functions as a learning or exploration tool.
- If a computer program returns a solution that is almost acceptable to the designer, the designer can get help on how to effect local changes. This helps avoid solving the entire problem again with only a slightly modified constraint.
- The designer can reliably predict the effects of changes i.e. in terms of which variables will be affected. One obvious extension is also to say how the variables will be affected.

The second class of algorithms actively search the network to identify portions that can be decoupled, allowing the designer to break a large problem into several smaller problems. If it is possible to separate 20 performance parameters of a large model into two independent groups of 10 parameters each, the reduction in complexity is appreciable. In most problems, we do not expect the algorithm to reveal such drastic simplifications. Obvious simplifications are likely to have been noticed by previous designers. In such cases, the search procedure can be modified to find "almost separable" groups of performance parameters, i.e. those with only one or two variables in common. This is helpful because, the problem can be decoupled by fixing the common inputs. The consequent loss of design freedom is more than compensated by the reduction in problem size.

- *Find-Separable-Portions.* It is sometimes possible to replace entire branches of a tree by a single, suitably-bounded variable. This is best illustrated through the following example. Let a performance parameter P be calculated as,

$$P = f1(A, B)$$

and the equations governing A and B be,

$$A = f2(C, D)$$
$$B = f3(E, F)$$

where bounds are available on C, D, E, F. This set initially involves optimizing over 6 variables but can be reduced to two subproblems with 4 and 2 variables respectively.

$$P = f1(A, B)$$
$$B = f3(E, F)$$

where bounds are available on A, E and F.
Once the value of A at the optimum is found, it is known to be achievable because of the bounds previously calculated. The values of the design variables C and D that

will yield the required value of A can then be determined. The advantage of this approach is that the number of equations to be handled during the optimization step can be reduced. However, the following condition must be satisfied.

Condition I: If the portion of the tree below A is replaced by a bounded variable with bounds A_{max} and A_{min}, then every value of A in that range must be achievable for some legal values of the input variables C and D, i.e. the equation f2(C, D) - x = 0, must have at least one solution satisfying $A_{min} \leq x \leq A_{max}$.

- *Find-Separable-n-pairs-of-Tree.* This is a generalization of the above concept to handle a situation where a search of the network does not result in removal of any separable nodes pairs. In this case, it is possible to find a groups of n nodes that satisfy the criterion of being independent. For example, consider a case where n=2

$$\begin{aligned} P &= f1(A, B, C) \\ A &= f2(D, E) \\ B &= f3(E, F) \\ C &= f4(G, H) \end{aligned}$$

Here, the influence of design variables D, E and F is limited to the two intermediate variables A and B. Hence the pair (A, B) forms a separable pair which can be optimized as a sub-problem. However, the higher dimensional analog of Condition I must hold. In our experience, this condition is hardly ever satisfied for groups of size greater than two.

2.2 Results

There is considerable potential for using the results of graph algorithms to simplify the optimization problems. The basic idea is to identify ways of decomposing the problem into smaller, simpler problems that can be handled without the need for human intervention. The graph algorithms described are much less computationally intensive than numerical optimization; they can be repeatedly run on very large networks in a few seconds. Hence it is economical to run them on every problem even if they do not always result in a substantial simplification. Secondly, the graph-based representation described turns out to be a good basis for building other applications because of the variety of information that can be easily stored and retrieved. The structure can also be easily maintained and modified.

3 Knowledge-Based Systems

Once the problem size has been reduced to the extent possible using the graph algorithms just described, the designer can use a Knowledge-Based System (KBS) to fix the values of the discrete design variables so that nonlinear programming techniques can then be used.

The knowledge-based approach is typically applied to problems where rigorous mathematical modeling is not possible[38]; math-programming techniques are more suited to problems where reliable mathematical models exist. We applied KBS technology to this problem because,

1. The complete model is too large for math programming-based solvers that we are aware of. This is due to a large number of integer variables[1].

2. Mixed integer nonlinear solvers that use branch-and-bound can be very slow and ineffective. We felt knowledge could be used to reduce the search space and hence achieve better performance.

3. For structural choices, rules are easier to read and interpret than the numeric constraints required by math solvers. Basic explanations for system actions can be generated more easily.

4. Designers are able to learn how to use the existing input-output system, pointing to the existence of heuristics for estimating the behavior of the network.

3.1 System Design

Automobiles are usually designed by perturbing an existing configuration and not by generating an entirely new one. We decided to use this methodology for the knowledge-based system also. We classify possible changes to the existing configuration as

1. Minor changes. e.g. changing the type of turbo mounted on an engine already equipped with a turbo.

2. Medium changes. e.g. adding a turbo to a normally aspirated engine.

3. Major changes. e.g. changing over from carburetion to throttle body fuel injection.

We assume that the magnitude of the difference between the target and current performance parameters can be used to decide on suitable modifications. For example, a 5 percent improvement is possible using a turbocharger but a 50 percent improvement is not. Another assumption is that the improvement required is monotonically related to the cost of making that improvement. For example, a 50 percent improvement in starting acceleration is likely to be more expensive than a 5 percent improvement. These numbers differ between performance parameters – in a particular situation, a 5 percent reduction in braking distance may be possible but a 5 percent reduction in 0-60 acceleration time may not. This knowledge has to be acquired from the designers.

[1]In subsection 4, we discuss how integer variables arise when modeling configuration choices

The characteristics of the current configuration are also relevant. A designer deals differently with an engine that is loaded with all the available performance enhancing accessories than with one that is a basic configuration. This presents an additional dimension for reasoning but we did not use this in our knowledge base.

Working with these assumptions and some preliminary information gathered from the users of the system we developed a prototype KBS which is described next.

3.2 System Implementation

The system first computes a qualitative descriptor for the change required in each performance parameter and uses these descriptors to select a course of action. Consider a case with three performance parameters P_1, P_2 and P_3.

The *change code* is a triplet as follows,

$$(C_1, C_2, C_3), \qquad C_1,\ C_2,\ C_3\ \epsilon\ (X, S, M, L)$$

where C_i is the change code corresponding to the performance parameter P_i while the letters X, S, M and L correspond to no change, small, moderate or large changes in a performance parameter.

When the system is presented with a set of targets, it computes the change code based on the difference between the existing configuration and the target and uses this code to decide which rules to fire. The following example illustrates this. Note that improvement of fuel efficiency and braking force means a numerical increase while for 0-40 acceleration time it means a numerical decrease.

Assume that the **Current Performance Parameter Values** are:
0-40 Acceleration Time(P1) = 10.0 s
Fuel Efficiency (P2) = 9 km/l
Braking Force(P3) = 7.5 m/s^2
and the **Target Performance Parameter Values** are:
0-40 Acceleration Time = 10.0 s
Fuel Efficiency = 9 km/l
Braking Force = 8.25 m/s^2

A Sample Change Code Computation Table (Braking Force) is

Change	Code
0.0-0.25 m/s^2	S
0.25-0.5 m/s^2	M
$>$ 0.5 m/s^2	L

Hence the change code in this example is (X, X, L). A sample rule that could then fire is,

If (Change Code is (X, X, L))
And (Current Configuration Has Drum Brakes)
Then (Add Disk Brakes)
And (Remove Drum Brakes)

The knowledge about magnitude of changes is implicitly encoded in the target code computation table whereas the knowledge about the configuration changes that can result in reaching the target for a performance parameter is encoded explicitly in the rules. For example, the rule just shown records the knowledge that disk brake performance is superior to drum brake performance.

The system was implemented in OPS5 and used as a preprocessor to a parametric optimization program. Details of how this system was integrated with other available tools are given in Section 5.

3.3 Results

Experiments with this system and discussions with expert designers yielded the following observations:

1. There are several problems inherent in developing qualitative descriptors of configuration and required target changes. For example, when discretizing a continuous range into approximate values we sacrifice precision; values which just barely fall into a particular class are made indistinguishable from values that fall squarely into the middle of the class. Hence, empirical studies using the acquired heuristic knowledge are an important part of the validation process.

2. Many designers feel that our model of their reasoning process, based on control codes, is not realistic. However, they are unable to come up with models of their own thinking. Some even question the very existence of simplified mental models. In our view, some type of qualitative model is used by the designers. Some improbable alternatives are that designers use pure generate-and-test(random search) or that they can visualize such a large nonlinear model. A more likely alternative is that they can learn simplified models that are valid only in very small regions of the space.

3. The idea of using target codes to control rule firings is not scalable. For N performance parameters and M qualitative ranges on each parameter, the potential number of change codes is M^N. If detailed knowledge on how to handle each type of change code existed, the system would then require M^N rules. For N=3 and M=4, the number is 64 and seems manageable. However, for N=19, M=4, the

number is 4^{19} and clearly unusable. An intuitive way to approach the solution of this problem is to reduce the number of target codes to a reasonable number. This may be done in several ways, such as decoupling the problem into smaller groups, applying clustering on target codes. The decoupling approach was discussed in Section 2.

4. The system uses a very "shallow" model. It adopts the simplistic view that the information content of the target codes is sufficient to achieve a satisfactory solution and hence suffers from fundamental limitations. It may be necessary to use a more detailed model to be able to reason effectively in a wide variety of solutions.

We have described how a knowledge-based system can be used to decide on values for the discrete valued variables in the problem. If the knowledge-based system can choose all the discrete variables, then the reduced problem is one of nonlinear programming (to decide the continuous variables). If some discrete variables cannot be fixed, then either the designer can fix them and use nonlinear programming or decide to use mixed integer nonlinear programming to determine the remaining discrete and all the continuous variables in a single step.

4 Math Programming

In this section we first describe some methods of solving the nonlinear programming problem that arises when the configuration (discrete) variables of the model are fixed and then describe how mixed integer nonlinear programming may be applied to the problem of determining both discrete and continuous variable values.

4.1 Nonlinear Programming

Background In the first stage of our investigation, we tried to reduce the problem to one of nonlinear programming by fixing all the discrete variables using the knowledge-based system and, if necessary, designer choices. The problems we experienced while formulating and solving the resulting nonlinear program were as follows,

1. Formulation of a suitable objective function: The problem requires optimization in the presence of multiple, conflicting objectives. We decided to reduce the problem to one of scalar optimization by creating a composite objective function.

   The first objective function tried was a symmetric quadratic penalty.

$$penalty = \sum_{i=1}^{n} (p_i - t_i)^2$$

$$p_i - performance\ parameters$$

$$t_i - target\ values$$

However, imposing an identical penalty for violation or over-satisfaction of a specification does not reflect actual practice.

To correct this problem, we used a smooth function created by the weighted addition of exponentials.

$$penalty = \sum_{i=1}^{n} (e^{n1*(p_i - t_i - d)} + e^{n2*(p_i - t_i - d)})$$

$$|n1| > |n2|$$
$$n1 > 0, n2 < 0$$
$$d = f(n1, n2)$$

(The constant term d is needed to ensure that the minimum penalty occurs for exact satisfaction of the target.)

2. Convergence: Important factors in improving the chances of convergence were,

   (a) Starting guesses. The characteristics of the current model of car were found to be excellent for this purpose, since most calculations are perturbation-based.

   (b) Bounds. The benefits of having good bounds are especially dramatic in multi-dimensional problems. For example, the actual legal range of engine displacement is 1500 to 3000 cc. However the actual change made by a designer is *never* more than ±200 cc because of other, unwritten, constraints. This information can be used to significant advantage. Other implicit bounds, such as those on quantities with physical meaning (pressures, temperatures etc.), should also be used to the maximum extent possible.

   (c) Infeasibility. The problem of infeasibility can be controlled with tight bounds and good starting guesses. This is especially important in this model, because of the large number of equality constraints.

   (d) Local Optima. Due to the nonlinear nature of the problem, there is no way of avoiding local optima. Automatic reformulation of the inputs to math programming solvers and the availability of fast computer hardware make it possible for designers to explore many local optima.

Implementation The model was implemented using the GAMS modeling language[7]. GAMS is a commercially available system that provides a uniform interface to a variety of linear and nonlinear solvers. The high-level nature of the language makes it possible to rapidly formulate and modify even complex models quite rapidly. We added another layer of abstraction to this language so that the designer could fix variables, choose configurations or make other changes in an interactive LISP environment. When optimization is required, the system automatically creates a GAMS input program incorporating all changes made since the last run, submits it to the solver, parses the results and displays

a summary of the solution on the screen. This high-level facility for running the optimization program is instrumental in maximizing the utilization of the math-programming facility because making minor changes and re-running the analysis is no longer a problem.

Results The important conclusion from this study is that even in large nonlinear programming problems it is possible, with care and patience, to achieve satisfactory results. This is not a startling conclusion. However, we are interested in studying how our experiences with this problem can help us tackle other similar ones. Specific points of interest are:

1. The role of iterative human tuning. In the course of solving such problems, repeated human intervention seems unavoidable. However, many tricks of the trade can be systematically recorded and automated. For example, proper scaling and generation of tight bounds.

2. Error checking. Correct interpretation of the results produced by an optimization program is crucial. In complex models there is always the possibility of modeling blunders which result in non-intuitive results (such as the wrong set of constraints being active at the optimum).

3. Local Decoupling. This is a useful phenomenon which we noticed that can be used if the search space can be very strongly bounded. It is best illustrated by an example. Consider two performance parameters P_1 and P_2. Each has its input variables C_{1i} and C_{2i} and there are three common variables x, y and z. The equations are

$$P1 = f_1(x, y, z) + f_2(C_{1i})$$
$$P2 = f_3(x, y, z, C_{2i})$$

   The two performance parameters are said to be locally decoupled if we are operating in the vicinity of a point (x_0, y_0, z_0) where $\partial f_1/\partial x$, $\partial f_1/\partial y$, $\partial f_{1z}/\partial z \simeq 0$, leading to the equations,

$$P1 = f_2(C_{1i}) + const$$
$$P2 = f_3(x, y, z, C_{2i})$$

   Hence, even though a graph search for global decoupling (described in Section 2) would find that P1 and P2 are coupled, they are locally decoupled and this can be used to simplify the equations.

4. Choice of optimization algorithm. The nonlinear solver used by our version of GAMS is MINOS5, which is based on solving a series of linearly constrained subproblems. This is not the best strategy in our case because most of our constraints are nonlinear. However, it is the most widely applied method for large-scale problems and will remain so until other methods such as sequential quadratic programming are widely applied to large problems.

In many cases, the system produced answers that were surprising enough that the users felt compelled to verify the system-generated values by hand. Some of the useful feedback received from users was as follows (along with our opinion of their feasibility).

1. Justifying the choice of an optimum. Designers felt more comfortable about the answers once they understood which constraints were active and why a particular solution was reached. However, interpreting a matrix of Lagrange multipliers is an unnecessary burden to place on designers. We have no real solution to this problem in the context of math programming systems.

2. Allowing interactive tuning of the coefficients corresponding to each performance parameter in the objective function would enable individuals to exert more precise control over the actions of the solver.

3. Generation of multiple solutions. Many users wanted the system to generate multiple solutions, i.e. several choices of inputs that satisfied the performance variables. They felt more confident if the choice of which of these solutions to use was left to them.

4.2 Mixed Integer Nonlinear Programming

Introduction Mixed integer nonlinear programming (MINLP) has the potential to handle problems in which discrete and continuous variables occur simultaneously. Most of the approaches to MINLP have been based on branch and bound search or variations thereof (see Section 7 for references).

Given the principles of mixed integer programming and the specific objectives of our project, the following points are relevant.

1. The scientific community does not have much experience with solving MINLPs and this technique is only gradually gaining acceptance.

2. Limits on the number of integer variables that can be modeled using existing MINLP solvers require that the number of configuration variables be first reduced by some other means.

3. The presence of local optima strongly affects algorithms that use nonlinear program solutions as bounds for branching. Global optimum-seeking methods such as simulated annealing[30] were rejected because they were too slow for this application.

Implementation Techniques for formulating if-then rules, Boolean constraints etc. as equivalent sets of algebraic constraints involving zero-one variables are well known in the math programming community[32]. For example, consider the if-then-else type constraint

V_{ij}	none	single	twin
sohc	0.10	0.20	0.17
dohc	0.30	0.50	0.46

Table 2: Sample Lookup Table

If x≥0 then y = f_1(x) else y = f_2(x)

This situation occurs often in engineering design when models are often valid below some threshold. This constraint can be expressed numerically as,

$$\begin{aligned} x_v \epsilon \{0,1\} \\ (x_v = 1 \Rightarrow x \geq 0 \quad and \quad x_v = 0 \Rightarrow x < 0) \\ y_1 = f_1(x) \\ y_2 = f_2(x) \\ y = x_v \star y_1 + (1 - x_v) \star y_2 \\ x = x_p \star x_v + x_n \star (1 - x_v), x_p \geq 0, x_n < 0 \end{aligned}$$

Our implementation used the DICOPT++ solver[37] in conjunction with the GAMS[7] modeling language. The DICOPT++ solver has the following limitations,

1. Only binary variables are allowed. Hence integer constraints must be reformulated as sets of binary variables. For example, the constraint Aϵ{1, 2, 3, 4, 5} is represented as
$$A = \sum_{n=1}^{5} A_n \star n, \sum_{n=1}^{5} A_n = 1, A_n \epsilon \{0,1\}.$$

2. All constraints involving the binary variables must be linear. However the most compact formulation of a problem is often nonlinear.

3. A maximum of 30 binary variables can be used. The number of binary variables used to formulate linear versions of structures such as tables often grows rapidly. Consider the case of a variable x defined as
$$x = f(turbo\_type, cam\_type)$$
Assume that an algebraic form is unavailable, and f is represented by the Table 3. One approach to encoding this is as follows.
Let T_1, T_2 and T_3 be zero-one variables representing choice of turbo types and C_1 and C_2 be binary variables representing choice of cam types. The following

numerical equations then achieve the effect of picking a single turbo and cam type and lookup of the appropriate value from the table.

$$\sum_{i=1}^{3} T_i = 1$$

$$\sum_{j=1}^{2} C_j = 1$$

$$x = \sum_i \sum_j T_i \star C_j \star V_{ij}$$

$$T_i, C_j \in (0,1)$$

This formulation uses 5 binary variables ($3T_i$s and $2C_j s$) and is nonlinear in the binary variables. A linear form can be achieved by associating a binary variable with each entry in the lookup table.

$$E_{ij},\ i = 1,2,3 \text{ and } j = 1,2 \text{ are the binary variables}$$

$$x = \sum_{i=1}^{3} \sum_{j=1}^{2} E_{ij} * V_{ij}$$

The linear formulation requires as many binary variables as there are possible configurations. A typical automobile design allows thousands of configurations and hence size is a major restriction.

Results

1. Most solutions were achieved not by branch-and-bound but by one of the preliminary heuristics applied by ZOOM, the linear solver used by our version of DICOPT++[2]. Moreover, experience with ZOOM has shown that it does not perform reliably on problems with more than 20 variables[37]. DICOPT++ has the potential capability of using other, more reliable linear solvers, but these were not tested.

2. To achieve solutions using branch-and-bound, compilation of as much constraint information as possible is essential. Performance can be further enhanced by information such as an intelligent variable ordering.

3. Handling the complexity of the input manually was a major problem. Checking large numbers of integer constraints is difficult. This is evident when looking at any MINLP formulation.

4. Interpretation and justification of the results was difficult. Not having an explanation for *why* a particular configuration was the best one did not inspire confidence in the system.

[2]Solution of the MILP problem is an intermediate step used by DICOPT++ to solve MINLPs

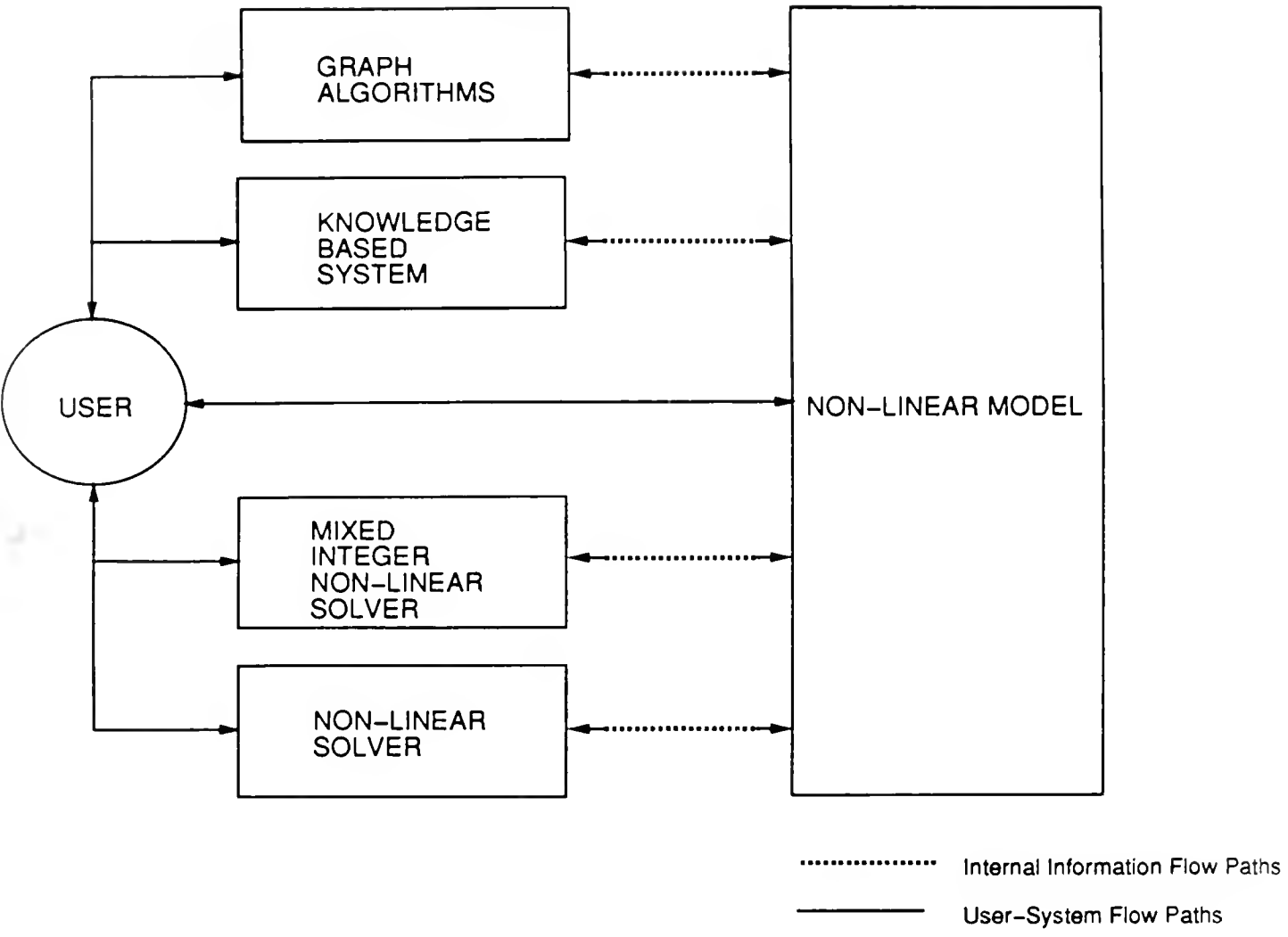

Figure 3: Integrated System for Design.

Hence, though theoretically the worst-case of the MINLP problem is intractable we can often achieve solutions if proper bounds and constraints are available. However, it is important to be constantly aware of the assumptions and limitations to which these solutions are subject.

5 Conclusion: Integrated Systems for Design

We have introduced a basic set of tools to solve the problem of parametric design involving configuration choices. We argue that a system that allows flexible, possibly simultaneous, access to this set of tools is superior to one which has the same tools as separate, non-communicating modules. Figure 3 shows an integrated system we built

which makes the three technologies described earlier available to a designer. The original solution methodology, using only the model, allowed only one path of information flow: from the user through the nonlinear model and back to the user. The new framework, while retaining the old mode of functioning, introduces many new paths that the design information can traverse. The automated tools available make it significantly easier to achieve a suitable design in a short time. More of the users' effort is directed into the actual problem solving because the procedural details are handled automatically by the system.

The computer-assisted design process now consists of the following steps,

1. Tools such as graph-based algorithms are used to identify new problem simplifications and decompositions.

2. On these reduced problems, a rule-based system provides advice on the discrete choices.

3. If all the discrete variables are fixed in Step 2, a non-linear program is automatically formulated and solved. Otherwise, a mixed-integer problem is formulated and solved.

4. At any stage, the designer can revert to the manual mode to achieve a solution. The system performs consistency checks in the background and flags violations.

5. The entire model, or parts of it, can be summarized or viewed and changes are easy to make. The graph-based representation allows automatic change propagation and correctness checking.

6. The system's expertise in any area can be smoothly complemented by the designer's knowledge. At any time, the designer can rapidly intervene, modify the algorithm and then return control to the system.

7. When tools fail, the system is still usable as it provides an indication of the reason for failure and it is easy for the designer to correct the problem and run the program again.

In this paper, we have attempted to examine the practical aspects of solving parametric design problems using available tools and extensions to these tools (graph representations and searching, new interfaces etc). We have proposed a novel framework for integrating new and existing tools that enables designers to solve problems from the real world. An example of using this methodology on a design problem drawn from industry has also been presented. We hope that these results will enable others facing problems of this nature to solve them more rapidly.

6 Acknowledgments

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by Nissan Motor Corporation and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K0124. The authors are grateful to the researchers at Nissan Central Scientific Laboratories, Yokohama, Japan for their intellectual contributions to this work.

7 Appendix 1: Related Work

7.1 Knowledge-Based Systems

Knowledge-based systems research is an established part of artificial intelligence (AI). The major issues include representation of knowledge, knowledge acquisition and implementation related issues such as environments, programming languages etc. Basic textbooks on AI discuss all these issues briefly [9, 27]. The classic papers on knowledge representation are available in collected form[33]. Similar collections of the important papers on pattern-directed inference systems[10] and rule-based systems[11] are also available. Several authors have discussed how to identify problems suitable for the application of expert system technology and give practical hints on implementing expert system tools [38]. Specific tools for implementing expert systems have also been widely discussed (see for example [5]).

The use of knowledge-based systems for design has been studied extensively and papers are available on the subject from widely differing domains, ranging from circuit design to machine element design [8, 25, 20, 34, 22, 24].

7.2 Mathematical Programming

There is a huge body of work on non-linear programming including several textbooks on the subject [12, 6]. These cover various aspects of optimization such as necessary and sufficient conditions for existence of extrema, use of computed and estimated derivatives and convergence. Papers describing specific optimization techniques also abound and are too numerous to list here. Important references can be found in the textbooks cited above. Integer linear programming techniques are covered in textbooks by Nemhauser and Wolsey[26] and by Garfinkel and Nemhauser[13].

There is also a considerable body of literature on the subject of optimization in design, including many textbooks[29, 31]. Interest in discrete optimization is more recent and till a few years back was limited to MILP solution. More recently, mixed integer- non-linear linear programming has become feasible due to the availability of cheap, high-speed computing. Much of the work in this area has been related to process plant synthesis for chemical engineering[39, 19, 1, 18] but mechanical design applications have been examined as well[23].

7.3 Graph Theory

Graph theory has many applications in science and engineering and has been studied extensively. Many excellent textbooks are available, for example the one by Henley[17]. The idea of representing constraints as graphs is standard in computer science (for ex-

ample, Gosling[15]) and attempts have been made to use graph-based algorithms to aid mechanical design[35, 16].

7.4 Combining Artificial Intelligence (AI) and Operations Research (OR)

Fundamentally, the fields of AI and OR both address the problem of search. However they approach the solution from different points of view. The AI community has been primarily concerned with symbolic reasoning in domains where good models do not exist while OR research has focused on the formulation and efficient solution of well-defined mathematical models. Conventionally, AI and OR have been independent academic areas with very little communication of methods or results. However, similarities have been noticed and the idea of combining AI and OR techniques has been suggested in the literature[14, 36]. In practice, many hybrid numeric-symbolic systems have been applied successfully [21, 3, 2, 4], Though, the consensus seems to be that most real-world design problems require well-integrated and flexible hybrid systems for their efficient solution, no systematic methodology seems to be evolving to guide the development of such systems.

8 Appendix 2: Sample Equations

The equations corresponding to Figure 2 are as follows,

$FE = CFE * (1 + (AP2 * DE2 + AP3 * DE3 + AP6 * DE6))$
$DE2 = VOL - VOLO$
$DE3 = R * IFO * IT4O/RO * IF * IT4$
$DE6 = NI - NIO$
$BRAKEG = (BFT + BRT)/WS$
$BFT = MBD * PMT * AF * RF/R$
If $(RBDISKTYPE \neq 0)$, **Then**
  If $(PMT \geq PMO)$, **Then**
    $BRT = MBD * DR1 * PMT + DR2 * PMO * AR * RR/R$
  Else
    $BRT = MBD * PMT * AR * RR/R$
If $(RBDRUMTYPE \neq 0)$, **Then**
  If $PMT \geq PMO$ **Then**
    $BRT = BEF * ((DR1 * PMT + DR2 * PMO) * ARD - DLOSS)/R$
  Else
    $BRT = BEF * PMT * ARD/R - DLOSS$
$WS = NPASS * ONEPASS + WEIGHT$

Note here that RBDISKTYPE and RBDRUMTYPE are discrete variables that decide whether the rear braking system is of the disk or the drum type. Hence, only one of these can be non null. Similarly, FBTYPE and MCYLTYPE choose the type of front brake type and the master cylinder type and hence must both always be non-null. This information must be represented explicitly in a MINLP model.

Some of the variables in the network are characteristics of a component and hence their value is decided by the choice of component. For example, AF is a friction factor that varies between brake systems. Hence, if say front brake of type 1 is chosen the corresponding value for AF must be looked up from a table using type 1 as the index into that table. The variables that depend on tables are as follows and are shown along with the appropriate index.

Variable	Index
AF	FBTYPE
RF	FBTYPE
AR	RDISKBTYPE
RR	RDISKBTYPE
DLOSS	RDRUMBTYPE
ARD	RDRUMBTYPE
BEF	RBDRUMTYPE
PMT	MCYLTYPE

It is also instructive to know what the type of each input variable is. This is shown in the following table.

Database	CFE, VOLO, RO, IFO, IT4O, NIO
Design	R, IF, IT4, NI,VOL (continuous) FBTYPE, RDISKBTYPE, RDRUMBTYPE, MCYLTYPE (discrete)
Constants	AP2,AP3,AP6, MBD, PMO, DR1, DR2, NPASS, ONEPASS, WEIGHT

References

[1] Duran M. A. and I. E. Grossman. A mixed-integer nonlinear programming algorithm for process systems synthesis. *AIChE Journal*, 32(4), April 1986.

[2] A. M. Agogino and A. S. Almgren. Symbolic computation in computer-aided optimal design. In J. S. Gero, editor, *Expert Systems in Computer-Aided Design*, pages 267–284, Amsterdam, 1987. North-Holland.

[3] S. Akagi, T. Tanaka, and H. Kubonishi. A hybrid-type expert system for the design of marine power plants using AI techniques and design automation method. In *Advances in Design Automation*. ASME, 1987.

[4] M. Balachandran and J. S. Gero. A knowledge-based approach to mathematical design modeling and optimization. *Engineering Optimization*, 12:91–115, 1987.

[5] D.Barstow et al . Languages and tools for knowledge engineering. In F. Hayes-Roth, D. Waterman, and D. Lenat, editors, *Building Expert Systems*. Addison-Wesley, 1983.

[6] M.S. Bazaraa and C.S. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, 1979.

[7] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A Users Guide*. The Scientific Press, 1988.

[8] D. C. Brown and B. Chandrasekaran. An approach to expert systems for mechanical design. Technical report, Artificial Intelligence Group, Department of Computer and Information Science, Ohio State University, Columbus, Ohio, 1983.

[9] E. Charniak, C.K. Riesbeck, and D. McDermott. *Artificial Intelligence Programming*. Erlbaum, 1980.

[10] Waterman D. and Hayes-Roth F. An overview of pattern-directed inference systems. In Waterman and Hayes-Roth, editors, *Pattern-Directed Inference Systems*. Academic Press, 1978.

[11] R. Davis and J. King. The origin of rule-based systems in AI. In B.G. Buchanan and E.H. Shortliffe, editors, *Rule-Based Expert Systems*. Addison-Wesley, 1984.

[12] Roger Fletcher. *Practical Methods of Optimization*. Wiley-Interscience, 1987.

[13] Garfinkel and G.L. Nemhauser. *Integer Programming*. John Wiley and Sons, New York, 1972.

[14] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[15] J. Gosling. *Algebraic Constraints.* PhD thesis, Department of Computer Science, Carnegie-Mellon University, 1983.

[16] M. D. Gross. *Design as Exploring Constraints.* PhD thesis, M.I.T., 1986.

[17] E.H. Henley and R.A. Williams. *Graph Theory in Modern Engineering: Computer aided design, control, optimization, reliability analysis.* Academic Press, New York, 1973.

[18] G. R. Kocis and I. E. Grossman. General optimization of nonconvex mixed integer nonlinear programming MINLP problems in process synthesis. *I and EC Research*, 27, 1988.

[19] G.R. Kocis and I.E. Grossman. Computation experience with DICOPT solving MINLP problems in process systems engineering. Technical Report EDRC-06-43-88, Carnegie Mellon University, 1988.

[20] A. Kott, G. Agin, and D. Fawcett. Configuration tree solver: A technology for automated design and configuration. In B. Ravani, editor, *Advances in Design Automation.* ASME, 1990.

[21] J.S. Kowalik, editor. *Coupling Symbolic and Numeric Computing in Expert Systems.* North Holland, 1986.

[22] K. Lien, G. Suzuki, and A.W. Westerberg. The role of expert systems techniques in design. *Chemical Engineering Science*, 42(5):1049–1071, 1987.

[23] H.T. Loh and P.Y. Papalambros. Computational implementation and texts of a sequential linearization algorithm for mixed discrete nonlinear design optimization. Technical report, Design Laboratory, University of Michigan, 1989.

[24] M.L. Maher, D. Sriram, and S.J. Fenves. Tools and techniques for knowledge based expert systems for engineering design. *Advanced Engineering Software*, 6(4), 1984.

[25] S. Mittal, C. Dym, and M. Morjaria. Pride: An expert system for the design of paper handling systems. In *Applications of Knowledge-Based Systems to Engineering Analysis and Design*, pages 99–116. American Society of Mechanical Engineers, 1985.

[26] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, 1988.

[27] N.J. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.

[28] C.H. Papadimitrou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity.* Prentice-Hall Inc, 1982.

[29] P. Y. Papalambros and D. J. Wilde. *Principles of Optimal Design: Modeling and Computation.* Cambridge University Press, Cambridge, England, 1988.

[30] W.H. et al Press. *Numerical Recipes.* Cambridge University Press, 1986.

[31] A. D. Radford and J. S. Gero. *Multicriteria Optimization in Architectural Design.* Academic Press, New York, 1985.

[32] R. Raman and I.E. Grossmann. Relation of logical inference and MILP modeling for chemical process synthesis. *Computers and Chemical Engineering*, (submitted), 1990.

[33] Brachman R.J. and H.J. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kaufman, 1985.

[34] Mittal S. and A. Araya. A knowledge-based framework for design. In *Proceedings of AAAI-1986*, 1989.

[35] D. Serrano. *Constraint Management in Conceptual Design.* PhD thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1987.

[36] H.A. Simon. Two heads are better than one: The collaboration between AI and OR. *Interfaces*, 17:8–15, 1987.

[37] J.V. Viswanathan and I. Grossman. *DICOPT++ Users Manual.* Engineering Design Research Center, Carnegie Mellon University, 1990.

[38] D.A. Waterman. *A Guide to Expert Systems.* Addison-Wesley, 1986.

[39] A.W. Westerberg. A review of process synthesis. Technical Report DRC-06-14-79, Carnegie Mellon University, 1979.